NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

REFINING DEEP LEARNING NEURAL NETWORKS FOR AUTONOMOUS VEHICLE NAVIGATION

by

Marcea M. Ascencio

March 2021

Thesis Advisor: Xiaoping Yun
Co-Advisor: James Calusdian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE REFINING DEEP LEARNING NEURAL NETWORKS FOR AUTONOMOUS VEHICLE NAVIGATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Marcea M. Ascencio	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Machine learning methods have recently increased in appearance in navigation and guidance applications by means of image classification. This thesis sought to advance the ongoing Electrical and Computer Engineering (ECE) Control Systems and Robotics Laboratory project in developing a system that will autonomously navigate across the Naval Postgraduate School (NPS) campus. In pursuit of providing a robust navigation and guidance solution to an autonomous robotic vehicle, a convolutional neural network (CNN) was trained to classify significant objects around NPS. In addition to increasing the number of objects that the neural network could classify, the network was also trained with varying image augmentation techniques applied to the training and validation images. A variety of tests were performed to evaluate the accuracy of the model when exposed to different objects and regions throughout the campus. The tests also included running the image classification model against images that were altered with crop, blur, rotation, and noise. The results demonstrated high classification accuracy and asserted that the output was robust when faced with poor image quality. This work established a strong baseline for more CNN output integration into the navigation and guidance solution of the robotic vehicle.

14. SUBJECT TERMS navigation, autonomy, machine learning, neural networks, CNN, robotics	**15. NUMBER OF PAGES** 69
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

REFINING DEEP LEARNING NEURAL NETWORKS FOR AUTONOMOUS VEHICLE NAVIGATION

Marcea M. Ascencio
Civilian, Department of the Air Force
BS, California State University, Long Beach, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2021

Approved by: Xiaoping Yun
Advisor

James Calusdian
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Machine learning methods have recently increased in appearance in navigation and guidance applications by means of image classification. This thesis sought to advance the ongoing Electrical and Computer Engineering (ECE) Control Systems and Robotics Laboratory project in developing a system that will autonomously navigate across the Naval Postgraduate School (NPS) campus. In pursuit of providing a robust navigation and guidance solution to an autonomous robotic vehicle, a convolutional neural network (CNN) was trained to classify significant objects around NPS. In addition to increasing the number of objects that the neural network could classify, the network was also trained with varying image augmentation techniques applied to the training and validation images. A variety of tests were performed to evaluate the accuracy of the model when exposed to different objects and regions throughout the campus. The tests also included running the image classification model against images that were altered with crop, blur, rotation, and noise. The results demonstrated high classification accuracy and asserted that the output was robust when faced with poor image quality. This work established a strong baseline for more CNN output integration into the navigation and guidance solution of the robotic vehicle.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AI	artificial intelligence
CNN	convolutional neural network
CPU	central processing unit
DAG	directed acyclic graph
DOD	Department of Defense
ECE	electrical and computer engineering
GUI	graphical user interface
LIDAR	light detection and ranging
P3-DX	Pioneer 3-deluxe
PC	personal computer
RAM	random-access memory
ReLU	rectified linearization unit
ROS	robot operating system
SLAM	simultaneous localization and mapping

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Xiaoping Yun and Dr. James Calusdian, for their guidance and support throughout this entire experience. Despite the challenges we have faced in the last year with distance learning, their unwillingness to sacrifice technical rigor and their mentorship and advice shaped the success of this thesis work. Thank you both for keeping me grounded and focused.

I would also like to thank a few professors: Dr. Brian Bingham for his excellent mentorship on autonomous robot control and graduate-level research methods, and Dr. Brij Agrawal and Dr. Jae Jun Kim for their fantastic deep learning course that really challenged my understanding of neural networks. These were some of the most influential lessons that I learned in my time here at NPS.

A final thank you to the ECE faculty, students, and community. I would have never expected my graduate program to be finished in a distance learning environment. In the face of these unprecedented times, there was never an instance where the technical quality of my studies faltered. In addition to continued academic excellence, I cannot thank everyone enough for their professionalism and efforts to keep the health and safety of everyone as their highest priority. We have been able to come out ahead because we did it together.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

As system sensor suites advance, complex algorithms are required to effectively utilize the incoming information for decisions such as navigation, guidance, and control. Machine learning models are used to interpret this incoming data and provide valuable outputs for these systems to use. Neural networks, a form of machine learning, are used for a variety of tasks, including object detection, image recognition, and classification. When incorporated into a sensor suite, the output of a neural network may be used to supplement existing navigation calculations or act as a sole input for navigation-based decisions. Since the neural network is onboard, the system does not have the same limitations that are commonly seen with radio or satellite navigation. As machine learning research has progressed, applications that utilize neural networks have become more robust and reliable, increasing the trustworthiness and usefulness of this technology.

As commercial technology has advanced towards autonomy to address the complexities of society, the Department of Defense (DOD) too is moving towards utilizing autonomy in warfighting technology. There is a demand for autonomous land, sea, and air vehicles to be developed to aid the warfighter in accomplishing varying mission types. The current DOD artificial intelligence (AI) strategy highlights focus areas such as "improving situational awareness and decision making, increasing the safety of operating equipment, implementing predictive maintenance and supply, and streamlining business processes." [1] By applying machine learning within military technology, certain decision-making tasks can be automated, leaving the operator to focus on higher-level decisions.

B. OBJECTIVE

In pursuit of advancing the existing Electrical and Computer Engineering (ECE) Control Systems and Robotics Laboratory project of developing a system that will navigate across the Naval Postgraduate School (NPS) campus, this research seeks to improve and refine the development and application of neural networks for use in the navigation and guidance of an autonomous system. The performance of the neural network must be able

to use images to classify significant objects and regions around the NPS campus. Additionally, there is a goal to create a robust neural network so that common issues such as cropped, blurred, or rotated images used by the autonomous system do not significantly alter the ability of the network to classify the desired objects or regions. This research will advance the overall project and is intended to be expanded upon in future work.

C. PREVIOUS WORK

The use of neural networks in image classification has been a topic of research for decades but has seen recent growth and popularity. Recently, convolutional neural networks (CNNs) have taken the lead in image classification applications. Prior research has led to a variety of algorithms, architectures, and implementations. This research is influenced by a variety of existing architectures and methods for image classification with neural networks. However, this thesis work on neural networks is tailored to the end goal of autonomously navigating a robot vehicle around the NPS campus.

Several ECE students have contributed to the overall effort of providing guidance and control to a robot vehicle so that it may autonomously traverse the NPS campus. This includes work done to provide waypoint following and object avoidance [2], route planning [3], terrain recognition [4], and even light detection and ranging (LIDAR) mapping for obstacle classification [5] to the autonomous robot vehicle. Most recently, however, this thesis furthers the work done in [6] by using CNNs for image classification applications in autonomous navigation. In Magee's work, a neural network was trained to identify an image of a goal location. The mobile robot performed autonomous navigation while simultaneously classifying images from its installed webcam. Upon identifying its goal by image classification, the vehicle stopped its motion. This thesis work seeks to build the database of identifiable objects around NPS, and most importantly refine the neural networks for a more robust output to be used by onboard navigation processes.

D. THESIS ORGANIZATION

The background, design, testing, and results of this thesis research are presented in the following three chapters. Chapter II contains a background of the hardware and software, image augmentation techniques, machine learning algorithms, and training setup

used in this research. A discussion of the experiments and results from the preliminary work, bench testing, and final classification testing are outlined in Chapter III. Finally, a conclusion of this research and listing of future areas of work are included in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In pursuit of developing an autonomous robot vehicle that can navigate around NPS, an array of hardware and software resources were utilized. The same robot vehicle that was used in the previous neural network research [6] was once again employed for this work to provide continuity and allow for more variation to be explored with the software development. Though the software used for the robot navigation algorithms was not altered, the neural network model was significantly improved upon. A brief overview of this hardware and software configuration is provided in this chapter. Additionally, this chapter elaborates on the techniques and algorithms used to refine the neural networks that were implemented in this research.

A. HARDWARE

The hardware used in this research consisted of a robot equipped with a mini personal computer (PC), various sensor systems, and a webcam. Additionally, an external computer was used for the machine learning algorithm design and development. The robot was an Omron Adept MobileRobots Pioneer 3-Deluxe (P3-DX) vehicle. Previous research of the onboard sensor systems (sonar, LIDAR, etc.) has been performed with this robot variant making it a prime candidate to execute the machine learning algorithms developed in this research. The onboard mini-PC was a SlimPro SP675P Mini PC equipped with an Intel Pentium Central Processing Unit (CPU) Model B950. The webcam used for input into the SlimPro Mini PC was a Microsoft Life Cam, with 720p resolution and a high-precision glass element lens. The external computer used was a Microsoft Surface Pro 6, with an Intel Core i5 processor and 8 GB of random-access memory (RAM). This external computer was used to develop the algorithms and models since the computations were processed quicker with this setup than the SlimPro Mini PC installed on the robot. All code was shared via server-based SharePoint prior to code execution that took place on the SlimPro Mini PC. The P3-DX is shown in Figure 1 with the mini-PC and webcam configuration.

Figure 1. P3-DX with Microsoft LifeCam Webcam

B. SOFTWARE

1. Robot Operating System (ROS)

Leading as the main collection of open-source software for robotic vehicles, Robot Operating System (ROS) provides a complete environment for applications in robotics. In previous research, comprehensive navigation scripts for the P3-DX with ROS applications and MATLAB, including waypoint following and simultaneous localization and mapping (SLAM) algorithms, were developed [6]. No ROS applications were modified or used for the purpose of this research.

2. MATLAB

MATLAB 2020b was the tool used to develop, train, and execute the machine learning networks on the P3-DX robot. All images captured via the webcam were pre-processed with MATLAB and then used as inputs into the machine learning functions. A variety of toolboxes were required to perform these tasks, which included the Deep Learning, Computer Vision, and Image Processing toolboxes. In addition, the Deep Network Designer graphical user interface (GUI) tool was used to perform a more visual-based assessment and design of the machine learning networks. Though the full design and training can be done within the Deep Network Design GUI, the preferred method was

completed within the MATLAB script editor. These MATLAB scripts are included in the Appendix.

C. IMAGE AUGMENTATION TECHNIQUES

Prior to being used for the training, validation, and testing of the machine learning networks, the images were pre-processed with MATLAB functions. This pre-processing altered the images to increase the diversity within each class. As seen with many other machine learning networks, image augmentation helps the network model increase generalizability, or increase the likelihood that the model will correctly classify unseen images when used in application. Training a machine learning network to correctly classify a large variance in image quality, such as images taken during high/low lighting conditions, in low contrast areas, or even with a noisy or blurred camera, can be challenging. The use of image augmentation techniques addresses this challenge and produces more resilient and robust networks.

1. Image Augmentation of the Overall Dataset

The overall dataset used to train, validate, and test this machine learning network contained 50% standard images acquired via the webcam and 50% altered images that were created with image augmentation techniques applied to the standard images. The MATLAB function “jitterColorHSV” was used to randomly alter the contrast, hue, saturation, and brightness of the images. This produced a larger dataset and increased the variance in the images. An example of this image augmentation technique is shown in Figure 2, where the altered image (right) was created by randomly skewing the contrast, hue, saturation, and brightness of the unaltered image (left).

Figure 2. Unaltered Image (Left) and Altered Image (Right)

2. Image Augmentation Specific to Training Dataset

The images in the overall dataset were randomly split into a training dataset, a validation dataset, and a testing dataset. A separate image augmentation technique was applied solely to the training dataset to alter the images further. It is common practice for most machine learning network development to highly augment the training images, but not the validation and test images. These added augmentation techniques included a randomized assortment of x-axis reflections, rotations, shearing, and translations. These types of augmentations are more extreme but increase the generalizability of the machine learning model when faced with sub-optimal conditions. An example of this image augmentation scheme is shown Figure 3. The torpedo image was reflected about the x-axis, rotated, and translated, and the building image was sheared.

Figure 3. Unaltered Images (Left) with Altered Images (Right)

3. Image Augmentation Scheme for Secondary Testing Dataset

Once the machine learning network was trained and validated with the MATLAB tools mentioned above, the network was then tested against a separate image dataset. This dataset consisted of the standard images and the images produced by the "jitterColorHSV" function that were set aside for testing. As a supplementary test, the images were then tested against a secondary dataset. Unlike the image augmentation scheme applied to the training dataset, the secondary test dataset had four subsets. Each of the subsets contained the same images but were separately altered by blurring, noise, rotation, and cropping augmentation techniques. This allowed for assessment of the network classification performance against each technique. This more isolated secondary testing method was inspired by [7] and proved more feasible to test a variety of conditions rather than waiting for the desired sub-optimal conditions around the school campus.

D. MACHINE LEARNING ALGORITHMS AND PROCESSES

As computer technology has advanced, research in the field of machine learning has capitalized on this growth and has flourished once again. Though machine learning has been used as a form of artificial intelligence for decades, recent advancements have utilized

the ability to apply more complex techniques to solve the most complex of problems. Machine learning can be succinctly summarized as the method in which a model is created from a set of training data.

There exist different methods of machine learning since varying applications require different models and have specific desired outputs. The main types can be broken into three categories: supervised learning, unsupervised learning, and reinforcement learning. The supervised learning method was applied in this research and will be discussed in this chapter. The other types, unsupervised learning and reinforcement learning, can be reviewed in "MATLAB Deep Learning" by Kim in [8]. Supervised learning is performed by training a model to arrive at a correct output by knowing the correct input. Common applications of supervised learning are classification and regression. As the name suggests, classification is the process of assigning inputs into classes. Regression applications, however, use the model to predict a value instead of a class.

The objective of this research is to use classification methods to provide navigation and guidance information to an autonomous robot. This is achieved by using supervised learning to develop a trained network based on labeled image data of landmarks, then applying the trained network to classify the desired landmarks. This application of classification is categorized as computer vision, or using a computer to interpret captured images. Different techniques for employing computer vision with machine learning techniques can be used, including a type of machine learning called deep learning.

In this chapter, the machine learning algorithms and processes that were used to create the image classifier employed by the robot are discussed. Additionally, a background of neural networks and convolutional neural networks is provided. Since the algorithms were modified iteratively over the course of this research and yielded a slightly different network with each iteration, the total variations amount to a large list. Therefore, only the final network architecture and training parameters are presented in the sections below. All final results discussed throughout this thesis are specifically based upon the output of this final trained network.

1. Neural Networks and Deep Learning

In supervised machine learning, a model is trained so it will produce a desired output given a known input. The model architecture used in this research is a neural network. Neural networks consist of nodes that are connected to each other with each connection defined by a weight value. The design of these neural networks is inspired by the neurons and connections of the human brain. Though not nearly as complex as the human brain, neural network architecture can be as simple or as complex as the system designer desires to achieve the end goal.

Neural networks can be broken down into three main types: a single-layer neural network, a multi-layer (shallow) neural network, and a deep neural network. The main difference between the three types is the increasing number of hidden layers in between the input and output layers of the network. The different levels of the neural network are called layers with each layer containing processing nodes. Neural networks have an input layer and an output layer; any layer in between is considered a *hidden layer*. A single-layer neural network has no hidden layers, a shallow neural network has one hidden layer, and a deep neural network has more than one hidden layer. In Figure 4, three examples of the neural network architecture are shown.

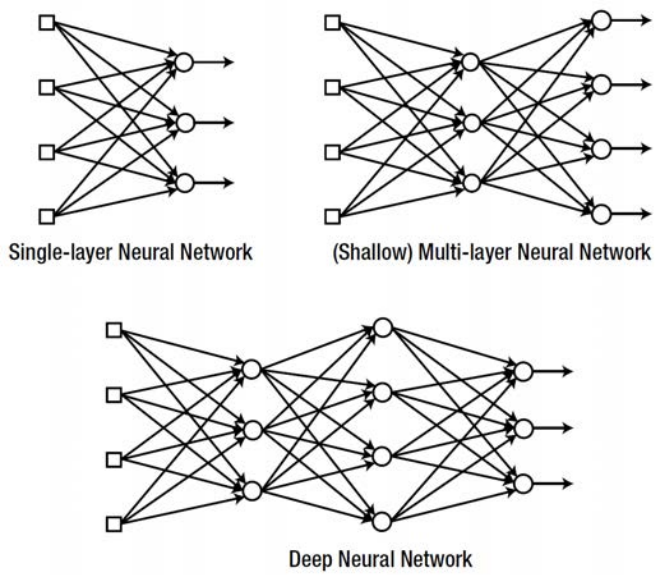

Figure 4. Different Types of Neural Network Architectures. Source [8].

During network training, the weight values of the connections are updated to minimize the error of the network output with the correct output defined by the training data. Despite the similarities between neural network architectures, single-layer neural networks require a different training rule than multi-layer networks. Kim summarized the learning rule used for single-layer neural networks, called the Delta rule, as "If an input node contributes to the error of the output node, the weight between the two nodes is adjusted in proportion to the input value, x_j and the output error, e_i " [8]. The Delta rule is not applicable to multi-layer training, and instead it requires an updated training rule.

In order to train a multi-layer neural network, a training rule called back-propagation is used. This is similar to the Delta rule in that error is calculated at the output node. However, the back-propagation algorithm takes this error and back-propagates it through the hidden layers until it is at the first hidden layer in the network. Then in similar fashion to the Delta rule, the weights are updated for their respective layers given the respective error at each node. This process is repeated until the output error is minimized to the desired value.

The output error of a network is quantified by means of a cost function also known as a loss function. Typically, the error is proportional to the value of the loss function; a larger error yields a larger value of the loss function. There are several different types of loss functions used in neural networks based on the specific output needs of the model. For multi-class classification purposes like the objective of this research, the most popular loss function used is the cross-entropy loss function. Alternatively called the soft-max function, this loss function utilizes logarithms to quantify the error. This allows the output value to be proportional to the input value and producing a value between zero and one. The function itself varies slightly as it is based on the different activation functions used within the network. This cost function is used to derive a learning rule for the network that is applied during network training. The goal of the learning rule is to minimize error and reduce the likelihood of creating a model that lacks generalizability or results in overfitting.

Overall, the architecture of a neural network and the algorithms it uses can vary greatly. This allowance has facilitated the advancement of machine learning applications

in everyday tasks. As described above, the deep neural networks are the networks with a larger number of hidden layers between the input and output layers. This type of machine learning, known as deep learning, allows for more complex problems to be solved. As Khan cites, deep learning offers three advantages: simplicity in developing large networks from basic building blocks, scalability that allows for large datasets to be used, and the ability for domain transfer when applying the model to other datasets [9]. Deep learning has achieved great success with minor improvements in the algorithms over the last few decades. These improvements have led to the use of CNNs for image classification and recognition allowing computer vision to take root in a variety of modern applications.

2. Convolutional Neural Networks

Convolutional Neural Networks are a specific type of a deep neural network. The purpose of the overall design is to identify features within images and use those features as input to the deep neural network for classification. This added section within the architecture of the CNN is referred to as the feature extractor. The complexity of this feature extractor can vary from a simple architecture that precedes the neural network to a large number of layers that are embedded into the total architecture of the neural network. There is no single formula of what a CNN architecture should look like, but the simple building blocks are convolutional layers, nonlinear activation functions, pooling layers, and fully connected layers. The variation of these architectures and algorithms employed within these networks in pursuit of better model accuracy is the bulk of current research.

The key feature of a CNN is the existence of convolution layers that are responsible for performing the convolution operations that are later fed into other layers for processing. These layers contain convolutional filters, which are 2-dimensional matrices comprised of discrete numbers. As the network is trained, these discrete numbers are altered for network optimization. To apply the convolutional filter to an input, the size of the filter and the stride length must be defined. As Kim states, “the convolution operation is the sum of the products of the elements that are located on the same position of the two matrices.” [8] An example of a convolution operation is shown in Figure 5. More detail can be reviewed in the texts by Kim [8] and Khan [9].

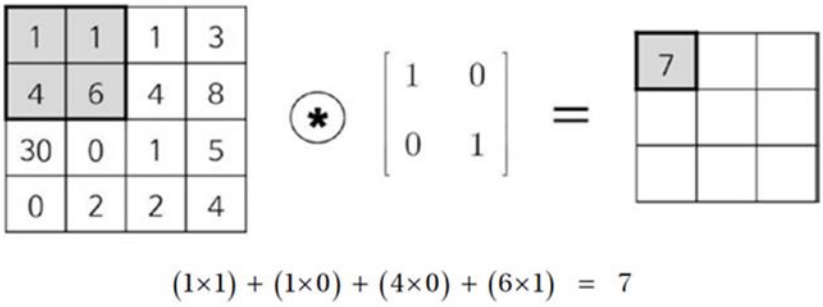

Figure 5. Convolution of a 2×2 Convolutional Filter and a 4×4 Input. Source [8].

The output of a convolutional filter is then processed through an activation function. There are many options for an activation function, the popular ones being a rectified linearization unit (ReLU) function, a sigmoid function, and the tanh function. More information for each function is presented in "A Guide to Convolutional Neural Networks for Computer Vision" by Khan in [9]. This nonlinear function modifies the output from the convolutional filters and changes it to fit within a small range of values. From these activation functions, a feature map is developed. The feature map is then down sampled within the pooling layers. Similar to convolutional filters, pooling filters are 2-dimensional matrices that are applied to the input and are defined by their size and stride. Convolutional neural networks typically employ either a max-value or a mean-value pooling function. Pooling is very useful when applied within networks as it reduces the computational load and prevents the network from overfitting [8]. The last building block is the fully connected layer, and as its name suggests, this layer is fully connected to all of the nodes in the previous layer. The fully connected layer takes the input, applies a weighted multiplication matrix, and introduces a bias value for the output. By combining all of the features learned in the previous network layers, the fully connected layer creates an output that is used to recognize the larger patterns across an input image [10]. By varying these building blocks, a CNN architecture can have an infinite number of possible configurations that provide the desired end result.

3. Network Architecture

There are several prominent CNN models that have garnered recent attention for applications in computer vision. Though there are other CNNs that could have been applied, this research solely focused on utilizing the CNN model, GoogLeNet. This is due to the large success that was seen with previous research [6] and due to the promising accuracy and efficiency as demonstrated in the ILSVRC 2014 Classification Challenge, in which GoogLeNet scored in the first place with a top-5 error of 6.67% [11]. By use of a method called transfer learning, GoogLeNet acted as the foundation for the CNN model used for all classification work performed in this research.

GoogLeNet is considered a directed acyclic graph (DAG) network, in which the architecture is typically more complex with multiple input and multiple output layers. Most notably, the architecture of GoogLeNet employs inception modules within the network. The central idea here is to place all the basic processing blocks (which occur in a regular sequential convolutional network) in parallel and combine their output feature representations [9]. An inception module is shown in Figure 6. By applying these modules throughout the architecture, complexity can significantly increase and cause a computational blow-up. By implementing a fully connected layer, the depth of the network can be reduced and lead to enhanced performance without potential for computational blow-up. Altogether, GoogLeNet shows a reduction in parameters from other CNN models such as AlexNet, and VGGnet, as well as a smaller memory footprint, a better efficiency, and a high accuracy [9].

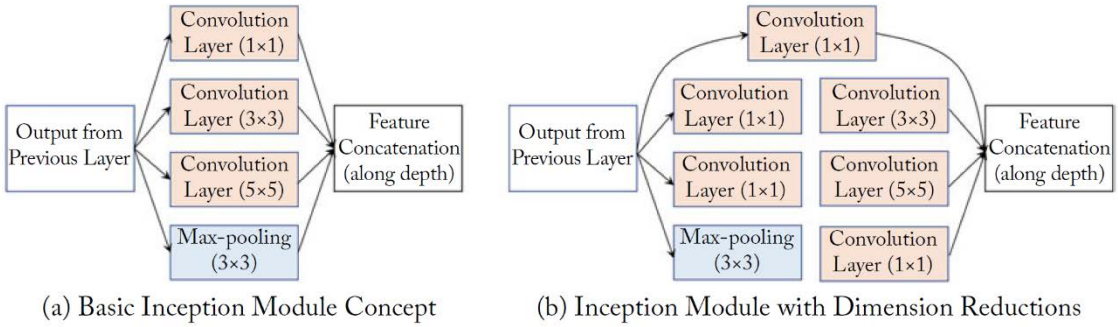

Figure 6. GoogLeNet Inception Module. Source [9].

The GoogLeNet CNN model was adapted for this research by applying a transfer learning process. This process varies for different types of CNNs and can be applied by manipulating different layers within the network depending on the desired network architecture. In the simplest method, GoogLeNet can be used for other classification tasks by replacing two of the last few layers within the network. This process includes replacing the existing fully connected layer with a new fully connected layer that has an output size equal to the number of classes for the classification task. Additionally, a new output classification layer is created. Once the new architecture is complete, MATLAB is used to perform training on the network based on the desired input dataset. This allows the new network to reap the benefits of the backbone CNN design. The GoogLeNet architecture and the layers that were replaced are shown in Figure 7.

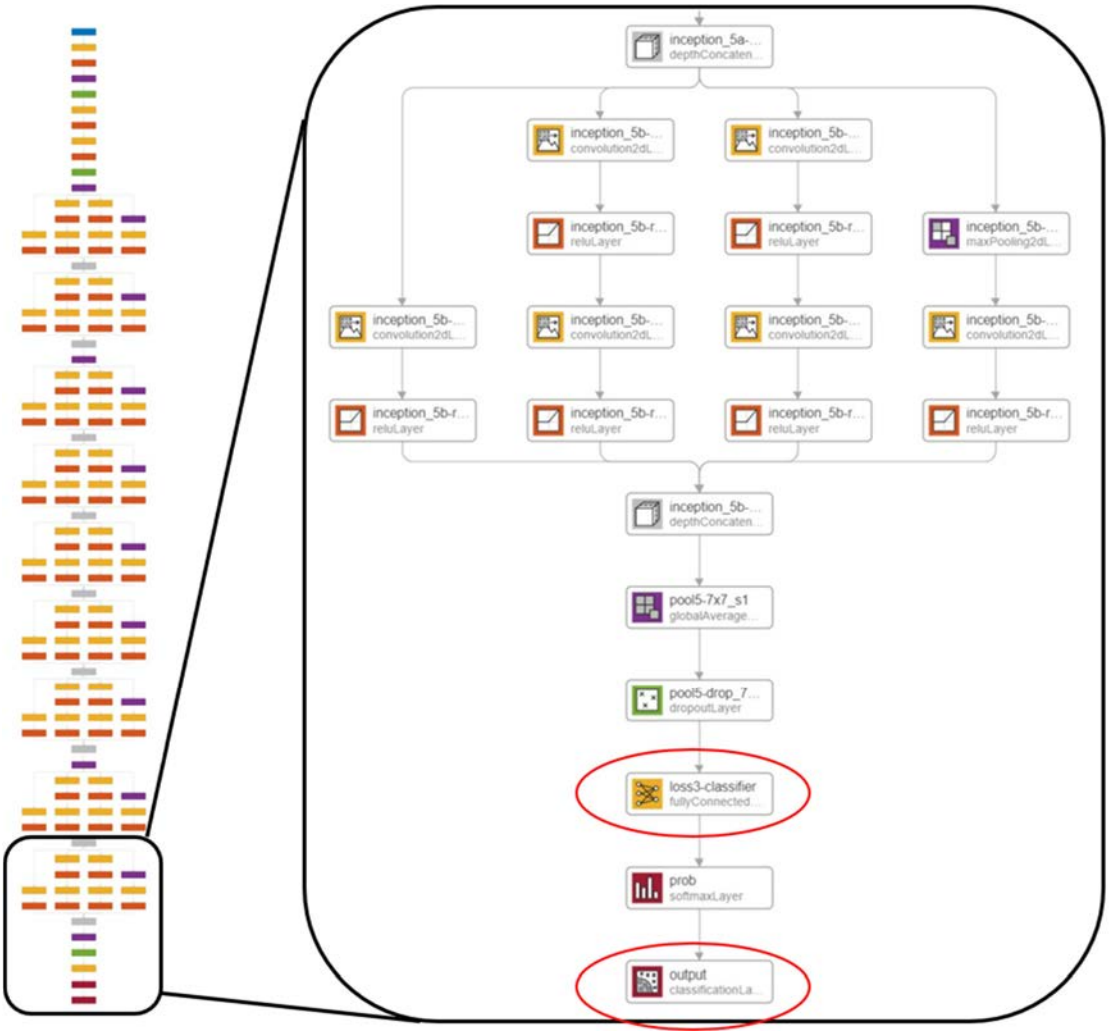

Figure 7. Simplified GoogLeNet Architecture (Left) With Layers That Were Replaced for Transfer Learning (Right) Circled in Red

4. Training Setup

The method in which a network is trained can vary and is dependent on the type of tools used. MATLAB offers a wide variety of toolboxes and functions that allow for specific tailoring within network training. These training options are typically altered in an iterative process of network training as the end goal of the network training is to produce the highest accuracy and lowest training time while preventing overfitting. MATLAB also provides a training plot that shows the network accuracy throughout training, as well as the progression of the training loss as calculated by the loss function discussed above. These plots are used to inform the network designer what types of training options need to be

altered to optimize the overall performance. Since the accuracy and speed of network training can be affected by more than just training options, there is not one single way to always optimize the training. However, MATLAB provides a simplified flowchart that is used in conjunction with the training accuracy and training loss plot to determine the suggested next step for bettering the network training, as shown in Figure 8.

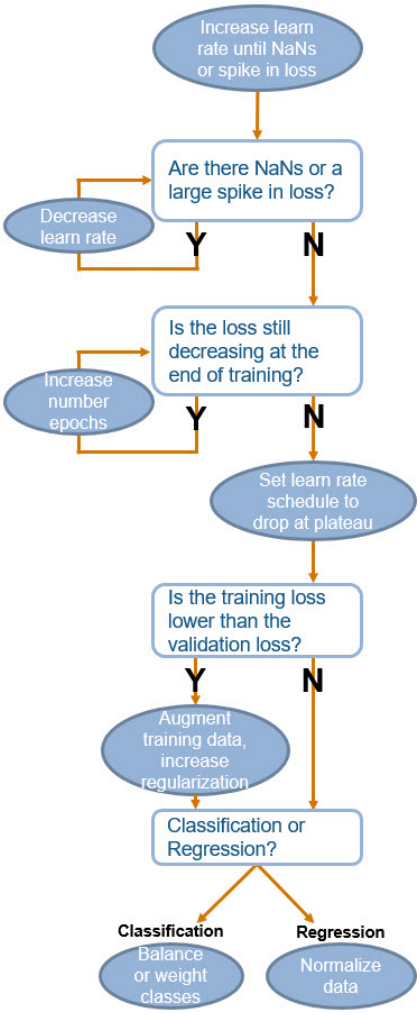

Figure 8. MATLAB Network Training Flowchart. Source [12].

After creating a diversified training set of all object classes, the 1900 images were randomly split into the following ratios: 70% for training, 20% for validation, and 10% for testing. An image augmentation code was applied to the training dataset to increase

diversity in the images in pursuit of increasing generalization of the model. The training options listed in Table 1 were used for the final network training.

Table 1. Training Options for Final Network

Training Option	Value/Setting
Momentum	0.9
Initial Learn Rate	1e-3
Learn Rate Schedule	None
Learn Rate Drop Factor	0.1
Learn Rate Drop Period	10
L2 Regularization	1e-4
Gradient Threshold Method	L_2 norm
Max Epochs	20
Mini Batch Size	40
Validation Frequency	15
Validation Patience	Infinite
Shuffle	Once
Sequence Length	Longest
Sequence Padding Value	0
Sequence Padding Direction	Right
Dispatch in Background	0
Reset Input Normalization	1

5. Training Results

The training and validation images, the neural network, and the training options were given as input to the MATLAB function “trainNetwork” and the network training was performed. The process took over 70 minutes to train the network, which involved updating the network weights to increase accuracy and minimize the loss. The final accuracy of 97.91% was achieved with a loss of 0.08. The plot in Figure 9 is the accuracy and loss as the training progressed. In the network training phase, these plots give insight as to whether a network is achieving its full potential for increasing accuracy and minimizing loss. Additionally, a network designer can identify if there is overfitting by ensuring that the validation accuracy is not less accurate than the training accuracy. Besides

testing the network against the test dataset, the training progress chart can give a significant amount of information regarding expected network performance.

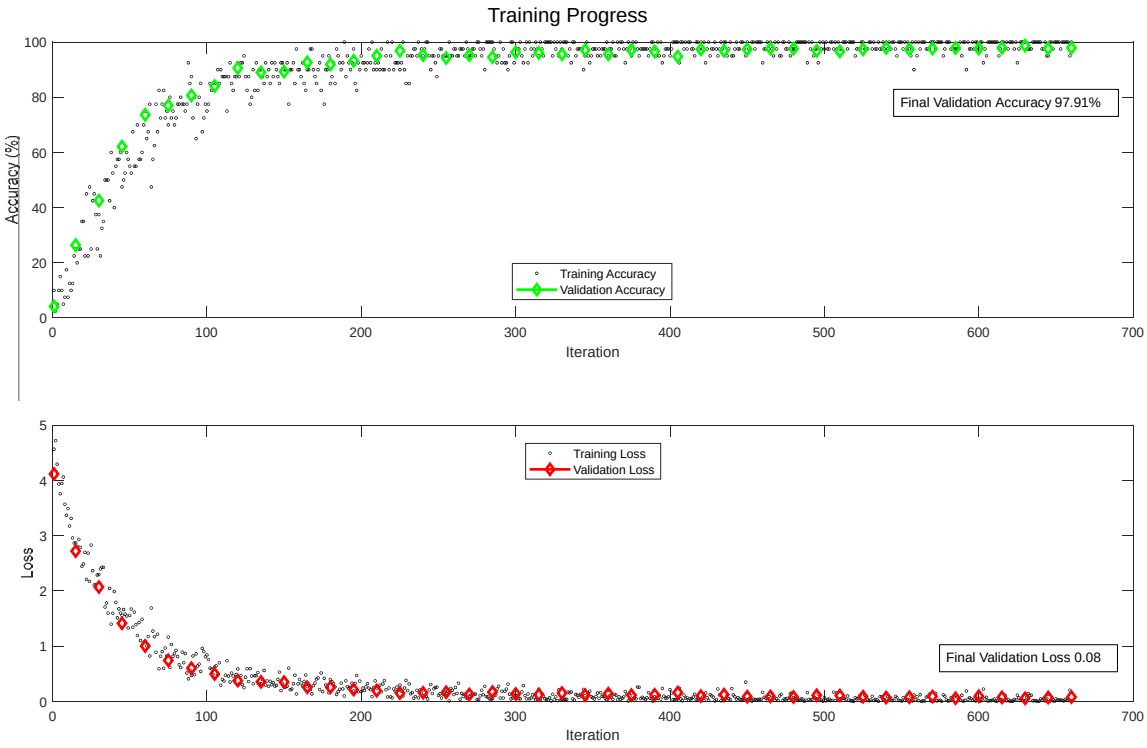

Figure 9. Training Progress Plot with Accuracy (Top) and Loss (Bottom)

III. EXPERIMENTS AND RESULTS

As the neural network was developed, different types of testing were performed and provided meaningful results that were used for network tuning and design. Prior to developing the final neural network model, some preliminary testing was conducted. This preliminary work explored the effects of image augmentation techniques on the network performance, which was later used in the creation of the training, validation, and test image datasets used for the final neural network design. Once the final neural network model had been developed, a few quantitative and qualitative tests were performed. As a final exercise, the neural network was tested while running on the robot vehicle hardware and mini-PC. An overview of each test setup and results are discussed in this chapter.

A. TEST SETUP

1. Preliminary Test – Image Augmentation Techniques

It is common practice to apply image augmentation techniques to the training image dataset used by the neural network. This idea was expanded upon by applying the image augmentation techniques to the validation dataset as well. Three different neural networks were trained from three different training datasets and then tested against the same test image dataset for comparison in classification robustness. The goal was to identify an office chair, independent of physical attributes such as size, design, shape, or color. The neural network was trained to produce only two separate outputs, a “chair” or “other.”

The first network was trained and validated from a variety of images that had no image augmentation techniques. The second network was the complete opposite in that all of the images used for training and validation were augmented. The third network combined the training image datasets from the first two networks, creating a dataset twice as large that contained unaltered and altered images. Image augmentation techniques included color adjustments, cropping, rotations, and blurring. An example of unaltered and altered images used in these networks are shown in Figure 10.

Figure 10. Unaltered Image (Left) and Altered Image (Right)

Each neural network demonstrated quick and successful training in MATLAB and did not require iterative attempts to refine the final validation accuracy. The training progress charts for the unaltered image network are shown in Figure 11, the training progress for the altered image network is shown in Figure 12, and the training progress chart for the combined image network is shown in Figure 13. As observed from these training progress plots, the training of each network took less than one minute and produced a 100% final validation accuracy and near-zero loss by the end of the final iteration. Each of these networks were tested against the same test image dataset, which was comprised of 109 unaltered images of object class “chair” and “other.”

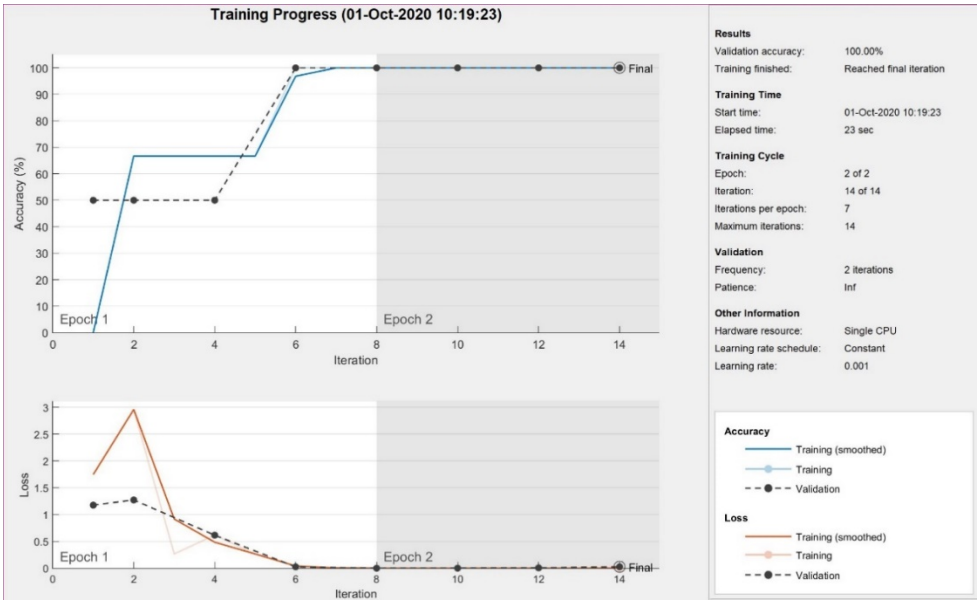

Figure 11. Training Progress Plot for Unaltered Image Network

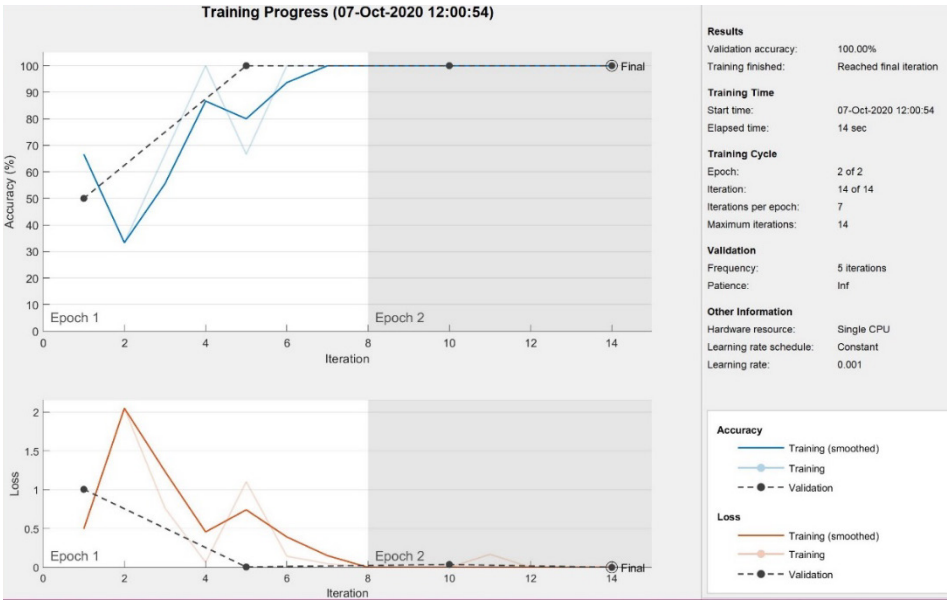

Figure 12. Training Progress Plot for Altered Image Network

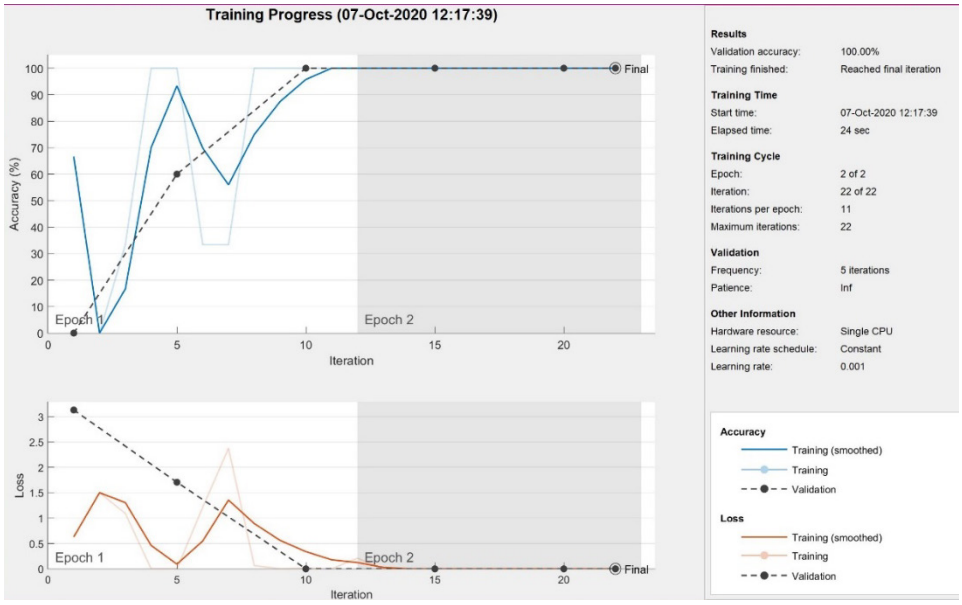

Figure 13. Training Progress Plot for Combined Image Network

2. Final Network Test

a. Test Setup for Final Trained Network

The final network was trained to identify 26 different object classes around the NPS campus. A representative image from each object class is shown in Figure 14. As

mentioned in the previous chapter, an image dataset containing 1900 images was used in the network training with 70% allocated for training images, 20% allocated for validation images, and 10% for test images. The final network was tested by classifying the images in the test image dataset with MATLAB. The classification accuracy was calculated as the percentage in which the predicted class matched the true class of each image.

Figure 14. Representative Images of Each Object Class of the Final Network

In addition to static image testing, the network was tested by performing a visual test of the network while moving the Microsoft Life Cam around the NPS campus. This setup included the external laptop running the MATLAB code and network model with the Microsoft Life Cam attached to the external laptop. This was all mounted to a cart and pushed around the campus, providing real-time image classification to the MATLAB figure window. Of note, the webcam was placed in a similar height and orientation on the cart that matched the mounting height and orientation of the camera on the P3-DX robot vehicle. Though not quantifiable, this visual check provided valuable feedback about the performance of the network.

b. Secondary Test Setup for Final Trained Network

In pursuit of a more quantifiable answer for network performance across wide variation of input images, a secondary test was performed. The secondary test utilized four different augmentation techniques each applied to a separate, replicated set of test images. These augmentation techniques included blurring, noise, rotation, and cropping techniques. The final network was tested against each augmented test dataset, including an unaltered image test dataset to serve as a baseline comparison. The classification accuracy of predicted class to true class was calculated and used as a measure of network performance.

The blurring augmentation was accomplished by creating a test dataset of images pre-processed with added Gaussian blur. The MATLAB function “imgaussfilt” applied a Gaussian blur filter with standard deviation equal to three to each image. The noise augmentation technique was also produced via a MATLAB function “imnoise” and was applied to each image. The function added Gaussian white noise with a zero mean and variance of 0.01. The third test dataset consisted of rotated images. The MATLAB function “randomAffine2d” was used to apply a randomized rotation between ±25° to the images. Finally, the fourth test dataset contained images that were cropped. A randomized window no smaller than 70% of the original image size was selected and then cropped with the MATLAB function “imcrop.” An example of an image within each dataset is shown in Figure 15.

Figure 15. Example of an Unaltered Image (Top), Blurred Image (Middle Left), Noisy Image (Middle Right), Rotated Imaged (Bottom Left), and Cropped Image (Bottom Right) Used for The Secondary Test

3. Integration Testing – P3-DX with SlimPro Computer

The majority of the network training, validation, and testing took place on the external laptop. A simple integration test was conducted on the SlimPro computer, which is the onboard computer for the P3-DX robot vehicle. The P3-DX was powered on with the SlimPro actively running the MATLAB network classification loop and the webcam mounted on the front of the robot vehicle. The loop captured snapshots as the robot moved through the campus and performed network classification on each snapshot. The images were reviewed after the test to ensure that the entire setup was able to run the classification tasks by the network without power, processing, or data storage issues.

B. TEST RESULTS

1. Preliminary Test Results – Image Augmentation Techniques

The preliminary test revealed that each of the three networks performed differently when tested against the same dataset. As expected, the network trained with unaltered training images had the lowest classification accuracy, 75%. The network trained with the altered training images and the network trained with the combined altered and unaltered training images had the same classification accuracy, 88%. A summary of these accuracies is presented in Table 2. Though the classification accuracies were the same for the "altered" and "combined" networks, each produced different classification errors.

Table 2. Preliminary Testing Results

Network Training Image Type	Classification Accuracy
Unaltered	75.23%
Altered	88.07%
Combined	88.07%

Another method of evaluating the performance of a network in a classification task is to examine how many of each class was predicted incorrectly. This is can often reveal the similarities between classes, identify inconsistent weighting of classes from network training, and highlight other unexpected downstream effects from the training dataset and network training. MATLAB produces a chart to map these mismatches with a "confusion chart" that displays the true class versus the predicted class for each image in the test dataset. The chart is divided into a grid-like display, with the numbers displayed in the boxes representing the number of predictions that were made in that category. A perfect classification network would have numbers only occupying the diagonal boxes of the chart, showing that the predicted class always matched the true class. However, it is more likely to see a few incorrect predictions reflected as the numbers contained in the other boxes of the chart. The confusion chart for the "unaltered" network is shown in Figure 16, the confusion chart for the "altered" network is shown in Figure 17, and the confusion chart

for the “combined” network is shown in Figure 18. The “unaltered” network did not do as well in classifying the “chair” class, with the highest amount of 26 incorrect classifications. It is also observed from these charts that although the “altered” and “combined” networks produced the same accuracies, they did not have the same classification errors.

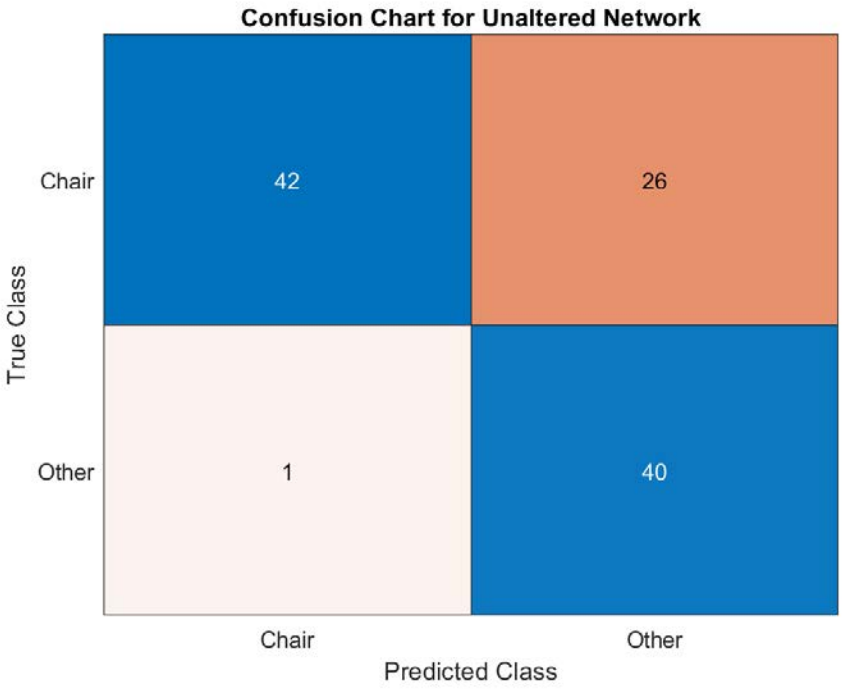

Figure 16. Confusion Chart for “Unaltered” Network

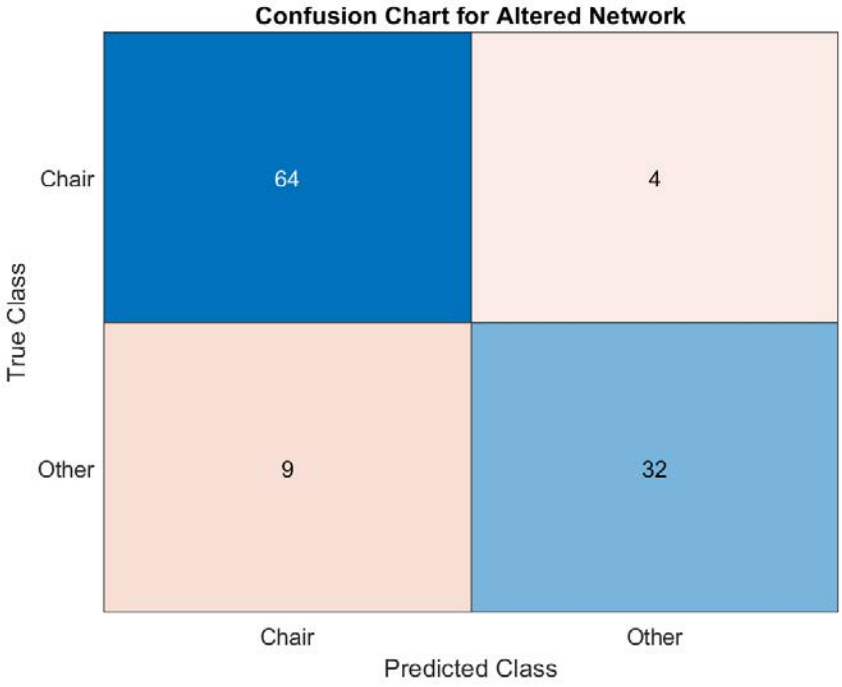

Figure 17. Confusion Chart for "Altered" Network

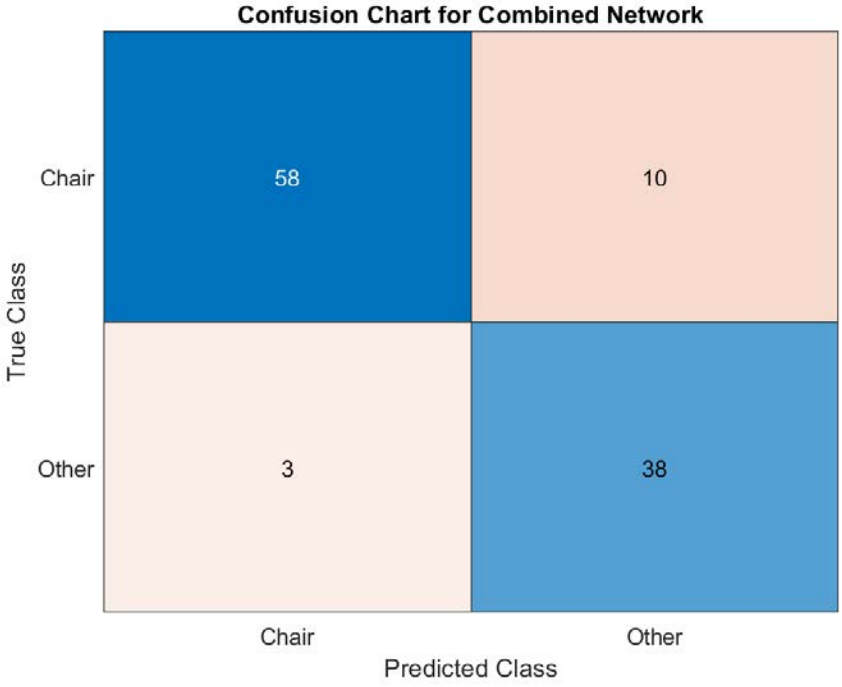

Figure 18. Confusion Chart for "Combined" Network

The results from this preliminary test reaffirmed the use of image augmentation techniques in both the training and validation images used for network training. Though there is not a discernable difference from this testing to suggest using an "altered" or a "combined" training and validation image dataset, the final network was developed by using a "combined" training and validation image dataset.

2. Final Network Test Results

a. Test Results for Final Trained Network

Upon network training completion, the final validation accuracy of the network was 97.91%. When the final network was tested against the test image dataset, the classification accuracy was 96.32%. Upon examination of the incorrectly classified images, it is evident that the true class visually resembles the incorrectly predicted class for most of the images. This is expected with the NPS landmarks since the school has very similar architecture and building colors across the campus. An example of this instance is shown in Figure 19 with the misclassified image and an example of the incorrectly identified class. The image on the left should have been classified as "Root Hall Underneath" but was instead classified as "Spanagel Back Door."

Figure 19. Incorrectly Classified Image with Predicted Class and Probability listed above (Left) and an Example of the Misclassified Class (Right)

Other similar misclassifications occurred and were made evident by examining the confusion chart, which is shown in Figure 20. For example, there was one instance of an image of the "Path Area" misclassified as "Courtyard." This is understandable as these

landmarks both span large areas outdoors, are not described by a single and definable feature, and are physically located near each other.

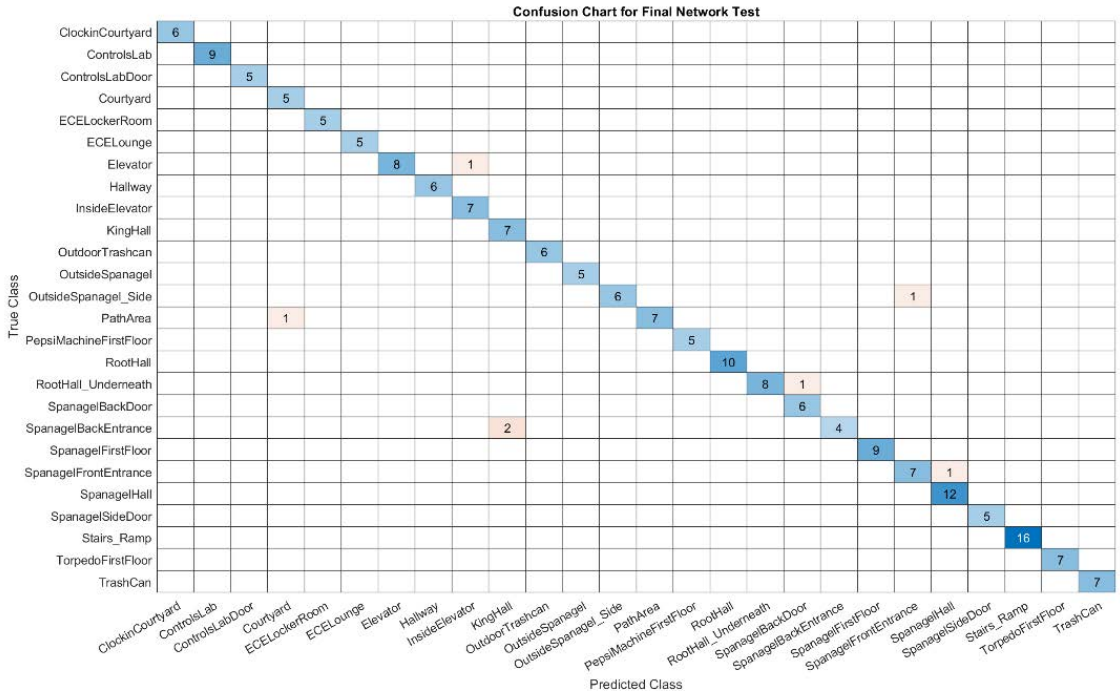

Figure 20. Confusion Chart for the Final Network Test

Another misclassification behavior was observed during the visual check. The network would correctly classify an object and when the image input shifted by moving the webcam, it would suddenly misclassify the object and then snap back to the correct classification. This quick misclassification was often seen with a lower prediction probability, typically a probability value of 85% or less. These observances suggest that if the output classification is to be used for the robot navigation solution, there should be hysteresis or timing thresholds applied to account for these sudden spikes in predicted classification.

b. Secondary Test Results for Final Trained Network

When tested against the four different augmented image test datasets, the classification accuracy of the network showed excellent results. The baseline accuracy was

determined by testing the network with an unaltered version of the test dataset. The baseline accuracy was 99.2%. The network classification accuracy for each test dataset was nearly the same, with the results summarized in Table 3. These results demonstrate that the network is robust and can still correctly identify the object class despite adverse conditions of the input image.

Table 3. Summary of Classification Accuracy for Each Augmented Test Dataset

Test Dataset Augmentation	Classification Accuracy	Difference from Baseline
Baseline	99.2%	-
Blur	99.3%	+ 0.1%
Noise	99.4%	+ 0.2%
Rotate	99.7%	+ 0.5%
Crop	99.4%	+ 0.2%

Though the results slightly differed, there were a few of the same images that were incorrectly classified across each dataset. The confusion chart for the baseline, blur, noise, rotate, and crop test datasets are shown in Figure 21, Figure 22, Figure 23, Figure 24, and Figure 25, respectively.

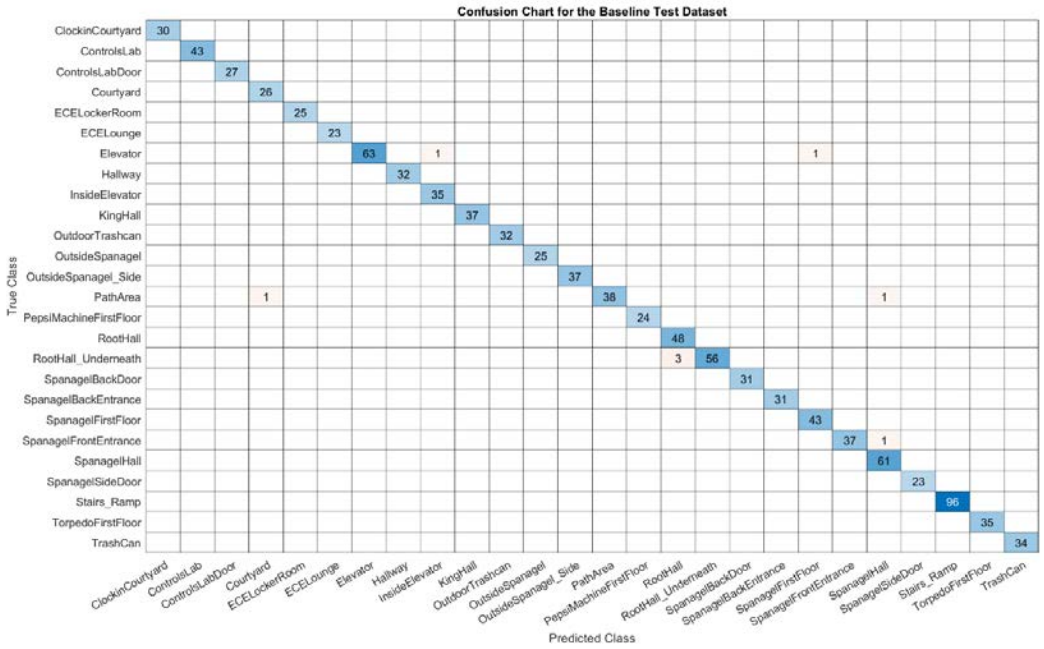

Figure 21. Confusion Chart for the Baseline Test Dataset

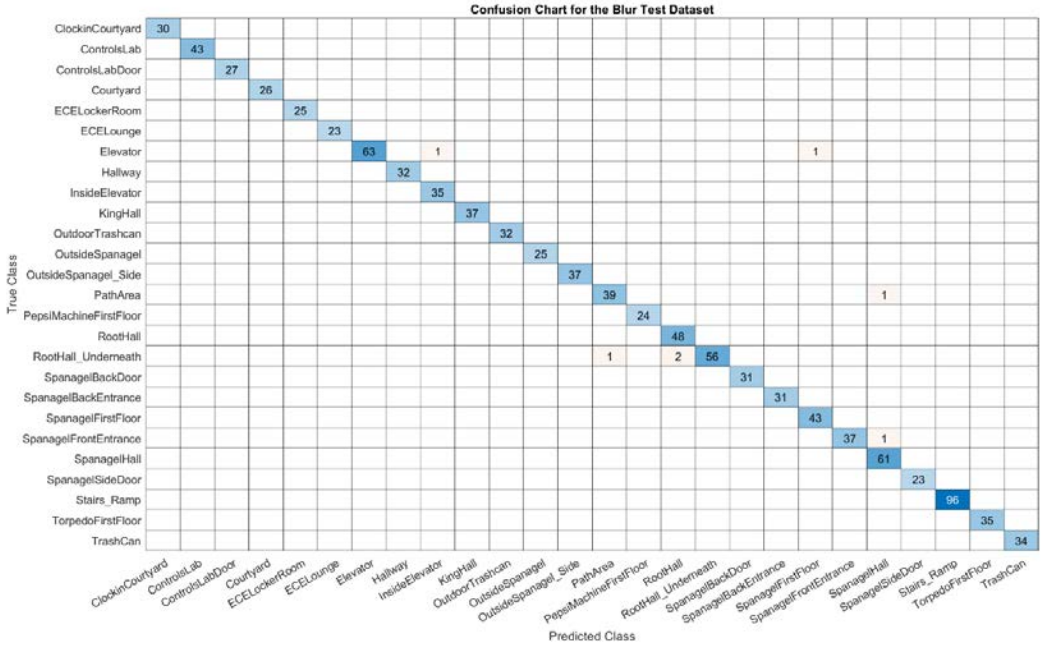

Figure 22. Confusion Chart for the Blur Test Dataset

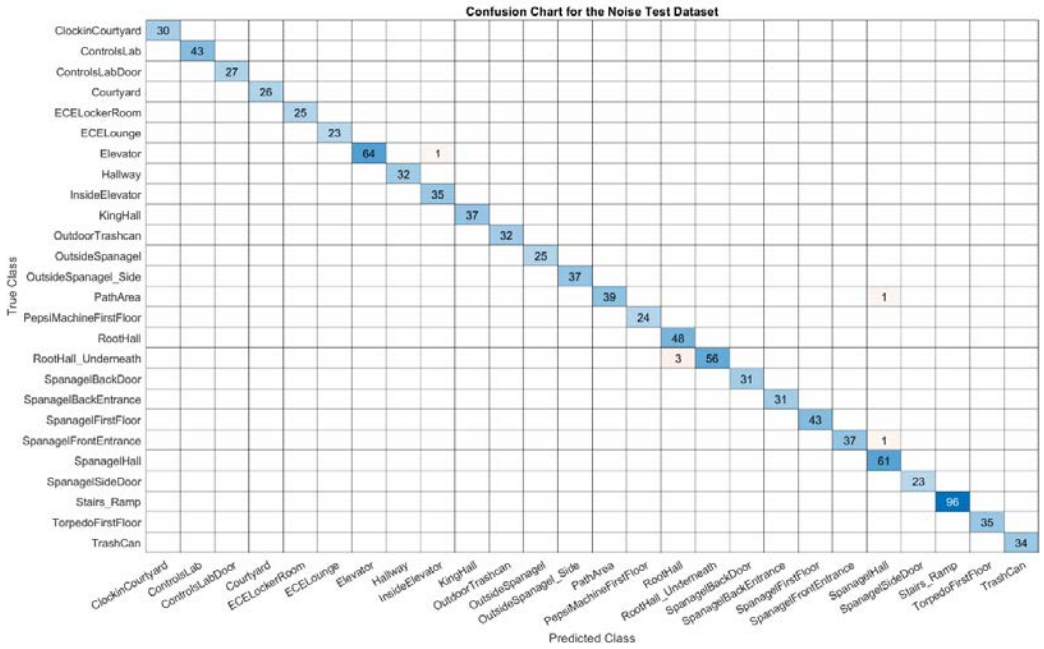

Figure 23. Confusion Chart for the Noise Test Dataset

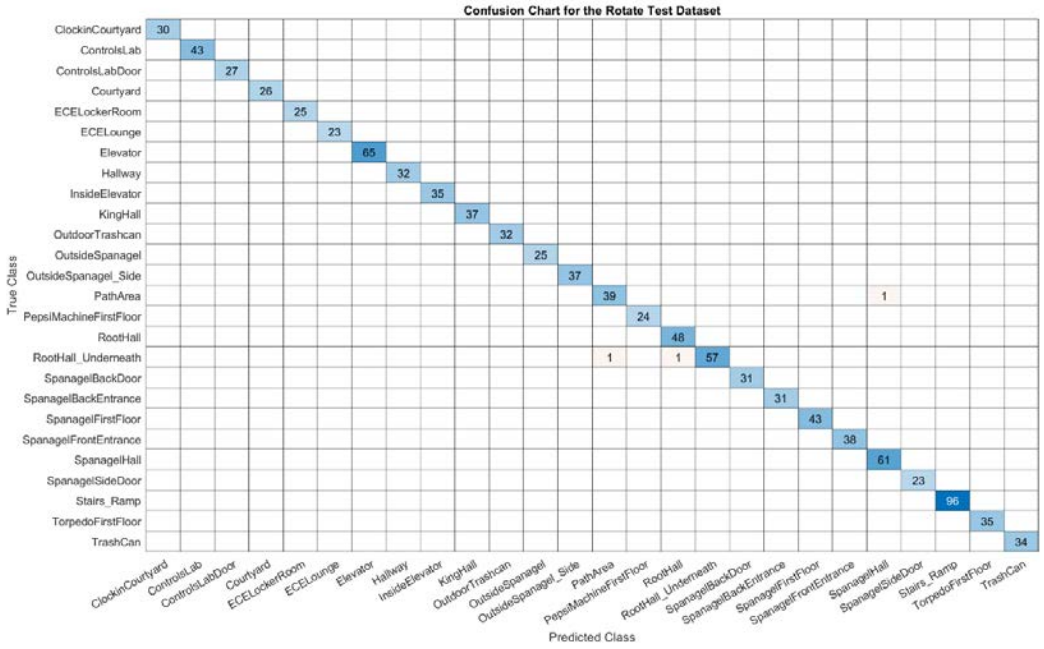

Figure 24. Confusion Chart for the Rotate Test Dataset

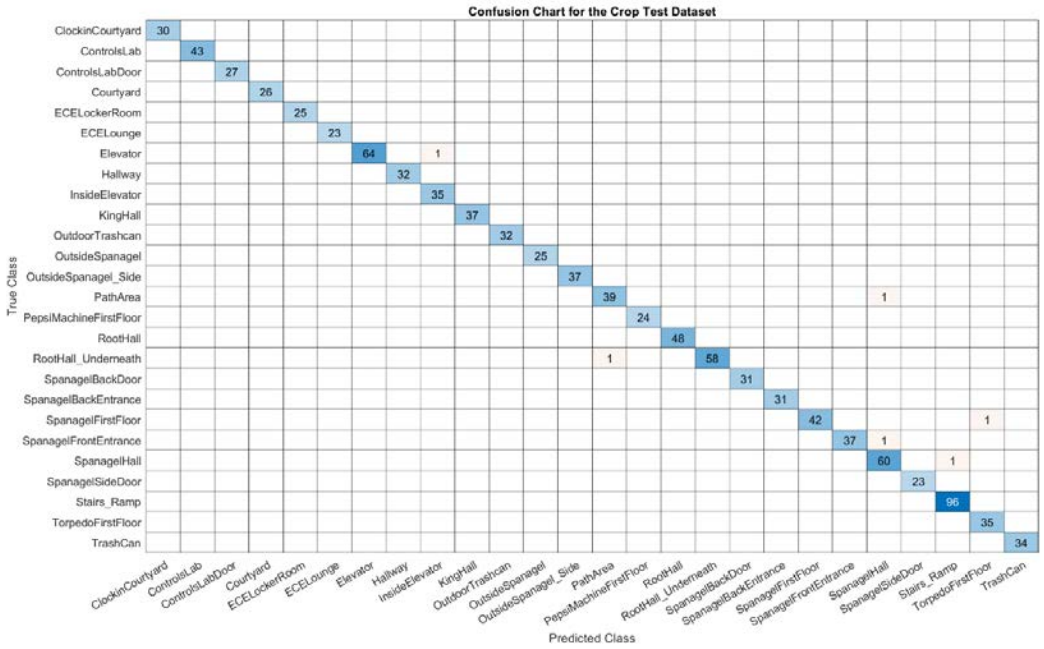

Figure 25. Confusion Chart for the Crop Test Dataset

3. Integration Test Results – P3-DX with SlimPro Computer

The integration test was successful, as the P3-DX robot vehicle was able to effectively classify the images captured during the test run around the campus. There were no obvious signs of processing degradation, power limitations, or data storage constraints. These test results will ensure a smoother transition when the output of the network classification is used within other robot navigation scripts.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONCLUSION

Machine learning applications have been a topic of research for decades but have seen a recent growth in popularity with the added capabilities provided by increased computing power. Deep learning, a form of machine learning, has been used for a variety of applications that require complex modeling. This research sought to apply deep learning techniques in pursuit of providing image classification for objects and regions surrounding the NPS campus as a means for navigation input to an autonomous robot vehicle. The neural network, or model employed in deep learning, was developed to recognize 26 objects around the campus. Additionally, by applying a variety of image augmentation techniques, this neural network showed robust performance when tested against blurred, noisy, rotated, and cropped images. The robot vehicle was able to utilize this neural network in a standalone configuration and proved that the robot vehicle can successfully utilize the outputs of the neural network. In this chapter, an assessment of the goals is provided as well as a discussion for future work.

A. ASSESSMENT OF GOALS

This research sought to improve and refine the application and training of neural networks used for navigation and guidance of an autonomous robot vehicle. The goal of this research was to further the previous ECE Control Systems and Robotics Laboratory research [6] by expanding the number of objects that the neural network could classify. Overall, the neural network developed in this research was able to identify 26 different objects around the NPS campus with a 97.91% validation accuracy. The neural network was also visually tested by running a live and continuous classification loop while moving the webcam around campus. This test also proved that the neural network had excellent performance of classifying the varying objects and regions through campus.

An additional goal of this research was to create a neural network that is robust and can perform classification tasks with input images that are blurred, noisy, rotated, or cropped. These types of altered inputs sought to reflect a more realistic scenario of real-world image acquisition. The neural network was tested against several test image datasets

that contained these image augmentations. The classification performance did not show degradation for any of the test image datasets with all of the classification accuracies above 99%. These results showed that the neural network can classify varying objects, despite having a substandard or highly altered input.

As a final check of the neural network, the autonomous robot vehicle hosted the classification tasks in a completely standalone configuration. This check was to ensure that the robot vehicle could successfully power and process the neural network without affecting the performance of object classification. There were no noticeable degradations to the performance of the network, and the robot vehicle was able to manage and store the output data successfully. In total, this work was successful in progressing the research of the ECE Control Systems and Robotics Laboratory in the pursuit of providing relevant navigation and guidance information for an autonomous robot vehicle.

B. FUTURE WORK

1. Incorporate Neural Network Outputs into Navigation Plan

The work performed in this research focused solely on developing the neural network performance but did not develop the integration of the neural network outputs into a robot vehicle navigation algorithm. The outputs of this neural network should be incorporated with other autonomous navigation code that is utilized by the robot vehicle. The results of this research have shown high fidelity in identifying objects around the campus and can be used in conjunction with other waypoint-based navigation techniques.

2. Expand the Number of Objects for Classification

Though the neural network was trained to identify 26 different objects and regions around the NPS campus, there are many more objects and regions that can be incorporated into the dataset. In addition to new objects, the expansion of this dataset should also include a greater variety of these objects with different lighting conditions and during different types of weather. Though different techniques were applied to test the robustness of the neural network, these techniques should be expanded upon. This increase in testing techniques includes creating more extreme test image datasets and developing new testing

methods. Ultimately, this will lead to a more generalized neural network that can traverse the entire NPS campus without issue.

3. Use CNNs for Object Detection and Localization

The output of the neural network developed in this research showed high accuracy in object classification but did not provide precise localization information. There is current research that utilizes CNNs in object detection algorithms that provide object classification and localization information from input images. This localization information is in the form of a bounding box that is placed around the object that is being classified. This type of deep learning is processing-intensive for both training and execution but can provide enhanced information to be used for navigation of an autonomous system.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. NETWORK TRAINING SCRIPT

```
%% NPS Classification Network Training
% Marcea Ascencio

% This script reads in the images for network training, creates the
% network architecture for transfer learning, applies required
% pre-processing, trains the network, and tests the network accuracy.

%% Read In Images to an Image Datastore
loc='C:\Users\marce\Pictures\NPS_Thesis';
imds=imageDatastore(loc,'IncludeSubfolders',true,'LabelSource','foldernames');
% montage(imds)
% Split data up into training, validation, and testing imds'
[trainds,valds,testds]=splitEachLabel(imds,0.7,0.2,0.1);

%% Load Network
net=googlenet;
%% Manipulate Layers for Transfer Learning
lgraph=layerGraph(net);
% Get number of classes from the training datastore:
numClasses=numel(categories(trainds.Labels));
% Create new fully connected layer and classification layer
newfc=fullyConnectedLayer(numClasses,"Name",'fc');
newcl=classificationLayer("Name",'cl');
% Display end of layers before replacing them
lgraph.Layers(end-2:end)
% Replace layers
lgraph=replaceLayer(lgraph,'loss3-classifier',newfc);
lgraph=replaceLayer(lgraph,'output',newcl);
% Display end of layers after replacement
lgraph.Layers(end-2:end)

%% Augment image datastores
% Get input size from the network
inputSize=net.Layers(1).InputSize

augmenter = imageDataAugmenter('FillValue',[256 256 256],...
    'RandXReflection',true,...
    'RandRotation',[-25 25],...
    'RandXShear',[-15 15],...
```

```
    'RandYShear',[-15 15],...
    'RandXTranslation',[-50 50],...
    'RandYTranslation',[-50 50]);

augtrainds=augmentedImageDatastore(inputSize(1:2),trainds,'
DataAugmentation',augmenter);
augvalds=augmentedImageDatastore(inputSize(1:2),valds);
augtestds=augmentedImageDatastore(inputSize(1:2),testds);

%% Training Options
miniBatchSize = 40;
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',20, ...
    'InitialLearnRate',1e-3, ...
    'ValidationData',augvalds, ...
    'ValidationFrequency',15, ...
    'Verbose',true, ...
    'Plots','training-progress',...
    'ExecutionEnvironment','auto');
[net,info] = trainNetwork(augtrainds,lgraph,options);

%% Test Network with Test image datastore
%Classify test images
[YPred,probs]=classify(net,augtestds);

acc=sum(YPred==testds.Labels)/numel(testds.Labels)
perprob=max(probs,[],2)*100;

figure
confusionchart(testds.Labels,YPred)
incorrectidx=find(YPred~=testds.Labels);
%
for k=1:length(incorrectidx)
    idx=incorrectidx(k);
    im=augtestds.Files{idx};
    im=imread(im);
    figure
    imshow(im)
    title(string(YPred(idx))+'  '+num2str(perprob(idx),4)  +
'%')
end
```

APPENDIX B. IMAGE AUGMENTATION SCRIPT

```
%% Image Augmentation Script
% Marcea Ascencio

% This script reads in images, applies desired augmentation
technique(s)
% and then saves them back into the desired destination
folder. Note, the
% loop can be commented/uncommented depending on the desired
augmentation.

clear,clc

%% Read Images
loc='C:\Users\marce\Pictures\SecondaryTest_Thesis\CleanImag
es\TrashCan';
dest='C:\Users\marce\Pictures\SecondaryTest_Thesis\Crop\Tra
shCan';
imds=imageDatastore(loc,'IncludeSubfolders',true,'LabelSour
ce','foldernames');
% figure
% montage(imds)

%% Parse Out Each Image and Augment

for k=1:length(imds.Files)
    k
    tempfilename=imds.Files{k};
    tempim=imread(tempfilename);

newname=strcat(dest,'\','crop_img_DayMonth',num2str(k),'.jp
g');

    % Reflection ------------------------------------------
---------------
    tformA=randomAffine2d('XReflection',true);
    outputView = affineOutputView(size(tempim),tformA);

tempim=imwarp(tempim,tformA,'OutputView',outputView,'FillVa
lues',[256 256 256]);
    disp('Image was possibly reflected')

    % Rotation --------------------------------------------
---------------
```

```
    tformB=randomAffine2d('Rotation',[-25 25]);
    outputView = affineOutputView(size(tempim),tformB);

tempimrot=imwarp(tempim,tformB,'OutputView',outputView,'Fil
lValues',[256 256 256]);
    disp('Image was Rotated')

    % Shear ---------------------------------------------
---------------
    tformC=randomAffine2d('XShear',[-15    15],'YShear',[-15
15]);
%     tformC=randomAffine2d('XShear',[-15 15]);
    outputView = affineOutputView(size(tempim),tformC);

tempim=imwarp(tempim,tformC,'OutputView',outputView,'FillVa
lues',[256 256 256]);
    disp('Image was Sheared')

    % Translation ---------------------------------------
---------------
    tformD=randomAffine2d('XTranslation',[-50
50],'YTranslation',[-50 50]);
%     tformD=randomAffine2d('XTranslation',[-50 50]);
    outputView = affineOutputView(size(tempim),tformD);

tempim=imwarp(tempim,tformD,'OutputView',outputView,'FillVa
lues',[256 256 256]);
    disp('Image was Translated')

    % rgb2gray ------------------------------------------
-------------------

%     rgbflag=0;
    tempim=rgb2gray(tempim);
    disp('Image was converted from RGB to Grayscale')
    rgbflag=1;

    % jitterColorHSV ------------------------------------
-------------------


tempim=jitterColorHSV(tempim,'Contrast',0.4,'Hue',0.1,'Satu
ration',0.2,'Brightness',0.3);
    disp('Image was altered with jitterColorHSV')
```

```
    % imnoise ---------------------------------------------
---------------

    tempim=imnoise(tempim,'gaussian');
    disp('Image was altered with imnoise')

    % imgaussfilt ------------------------------------------
---------------

    blurval=3;
    tempim=imgaussfilt(tempim,blurval);
    disp('Image was blurred')

    % Crop -------------------------------------------------
---------------
    h=size(tempim,1);
    w=size(tempim,2);
    percentcrop=0.3*rand;
    hcrop=round(h*(1-percentcrop));
    wcrop=round(w*(1-percentcrop));
    targetSize=[hcrop wcrop];
    win = centerCropWindow2d(size(tempim),targetSize);
    tempim = imcrop(tempim,win);

    % save image
    imwrite(tempim,newname)

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. LIVE TEST SCRIPT

```
%% Live Test Script
% Marcea Ascencio

% This script runs a live test by utilizing a trained network
and continous
% while loop to run classifications on the input images
captured by the
% webcam. The classification is displayed in the figure window
with the
% respective object class and probability. Note- make sure to
load network
% prior to executing this script.

%% Setup
% load webcam
camera=webcam('Microsoft® LifeCam Cinema(TM)');
% create input size variable for incoming images
inputSize=net.Layers(1).InputSize(1:2);
%figure handle
h = figure;

%% Execute
% continous loop will execute while figure is open
while ishandle(h)
    im = snapshot(camera);
    image(im)
    im=augmentedImageDatastore(inputSize(1:2),im);
    [label,score] = classify(net,im)
    title(string(label) + "," + num2str(100*max(score)) +
"%")
    drawnow
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Department of Defense, “Summary of the 2018 Department of Defense artificial intelligence strategy.” Washington DC, USA, 2019.

[2] C. S. Hargadine, “Mobile robot navigation and obstacle avoidance in unstructured outdoor environments,” M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2017. [Online]. Available: https://calhoun.nps.edu/handle/10945/56937

[3] M. R. Audette, “Interactive map making for route planning and obstacle avoidance in an unstructured outdoor environment,” M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: https://calhoun.nps.edu/handle/10945/60406

[4] C. Lebrun, “Vision-based terrain classification and learning to improve autonomous ground vehicle navigation in outdoor environments,” M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: https://calhoun.nps.edu/handle/10945/63474

[5] A. S. Miyakawa, “Autonomous ground vehicle low-profile obstacle avoidance using 2D LIDAR,” M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: https://calhoun.nps.edu/handle/10945/63486

[6] A. Magee, “Place-based navigation for autonomous vehicles with deep learning neural networks,” M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2019. [Online]. Available: https://calhoun.nps.edu/handle/10945/64012

[7] C. Liu, Y. Tao, J. Liang, K. Li, and Y. Chen, “Object detection based on YOLO network,” in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Dec. 2018, pp. 799–803, doi: 10.1109/ITOEC.2018.8740604.

[8] S. Kim, *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*. New York, NY, USA: Apress, 2017.

[9] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, “A guide to convolutional neural networks for computer vision,” *Synth. Lect. Comput. Vis.*, vol. 8, no. 1, pp. 1–207, Feb. 2018. [Online]. doi: 10.2200/S00822ED1V01Y201712COV015

[10] Mathworks, “Fully connected layer - MATLAB.” Accessed Feb. 09, 2021. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html

[11] C. Szegedy et al., “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. [Online]. doi: 10.1109/CVPR.2015.7298594

[12] Mathworks, Class Lecture, Topic: “Deep learning with MATLAB”, Mathworks, Online. https://www.mathworks.com/training-schedule/deep-learning-with-matlab.html

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California